Remarks

The present paper is in response to the non-Final Office Letter mailed in the above-referenced case on November 13, 2002, being the first action in the case. In the action the Examiner has rejected claim 4 under 35 U.S.C. 112, first paragraph, as containing subject matter which was not described in the specification in a way that would enable a person of ordinary skill to make or use the invention. The Examiner states that it is not explicitly clear what is the definition of "hardware call...return stack". Further, claim 4 stands rejected under 35 U.S.C. 112, second paragraph, as being indefinite, stating that "hardware call...return stack is an indefinite term".

Further to the above, claims 1-20 stand rejected under 35 U.S.C. 103(a) as being obvious over Song et al., US 6,061,711, hereinafter Song, in view of Bronte, US 6,061,709, hereinafter Bronte.

Applicant has thoroughly and carefully reviewed the references provided, especially the portions relied upon by the Examiner, and whether the references in those portions actually teach what the Examiner alleges that they do. Applicant has further reviewed the Examiner's statements and arguments in support of the rejections in light of the applicant's considerable knowledge in the relevant art. Applicant provided this analysis below, together with arguments traversing the Examiner's position on several points.

In regard to the section 112 rejections of claim 4, applicant points to the teaching in applicant's as-filed application, beginning on page 5 at line 23, and continuing to the end of the Background section. This portion of applicant's teaching is reproduced here, with Bold emphasis added:

"A cooperative system is similar to the pre-emptive system described above, but a context switch may only occur when and where the programmer has requested or allowed a context switch to occur. Since, when a subroutine or task is pre-empted in either type system, its *return addresses* are normally kept on a *stack*, the kernel must maintain a *separate stack for each task* so that when a task resumes, its subroutines are able to return back up the call tree. Context-sensitive information (e.g. the task's resume address and register values) is often also saved on the *task stack.*

As an example, a task T1 may call subroutines S1_1 and S1_2. Subroutine S1_1 may itself call subroutines S1_1_1 and S1_1_2, and subroutine S1_1_2 may call a subroutine S1_1_2_1. Task T1's call tree will look like:

```
T1
+------- S1_1
| +-------- S1_1_1
| +-------- S1_1_2
| +-------- S1_1_2_1
+------- S1_2
```

When a context switch occurs in subroutine S1_1_2_1 while task T1 is running, task T1's call...return stack will contain

```
top of stack -> return address in S1_1_2
return address in S1_1
return address in T1
```

This call...return stack is unique to task T1. When task T1 resumes after this context switch, each subroutine must successfully return to the position from where it was called in order for task T1 to continue executing properly. The kernel is responsible for managing all the stacks in the RTOS.

Architecture dictates the capabilities of a processor's stack. In a *general-purpose stack* (e.g. Motorola 68000 series microprocessors), the size of the stack is limited only by available RAM. Because the stack's contents are not constrained, a general-purpose stack can include subroutine return addresses. An RTOS allocates task stack memory based partially on the amount of memory required to store return addresses for the expected maximum call depth. As maximum call depth and/or the number of tasks in a system grows, so do its RAM requirements. RAM requirements per task for a typical RTOS range from hundreds of bytes to kilobytes or in some cases even megabytes per task. An RTOS may also allocate additional memory on a per-task basis for task-specific information, e.g. its priority and context-saving information.

Given the above descriptions, as current RTOS design presupposes the existence of one or more general-purpose stacks or similar memory blocks for the saving of return addresses, the RAM requirements of such an RTOS are a direct consequence of the RTOS' support for context switching at any call depth.

The typically very large RAM requirement for an RTOS supporting context switching at any call depth effectively preclude the use of an RTOS in a class of processors (usually microcontrollers) which have very little RAM, from tens to hundreds to perhaps as much as a few kilobytes of RAM. Such processors may have a general-purpose stack, or they may have a single *hardware call...return stack,* which provides for a restricted subroutine call

depth (often limited to 2-16 levels), with only return addresses stored on the stack. In either case there is too little memory to devote to individual task stacks.

Thus the memory-hungry nature of a conventional RTOS precludes use of such an operating systems with memory-challenged microcontrollers. What is therefore clearly needed is an RTOS that may be used with such controllers, which would significantly increase the range of uses for such controllers, and allow system design for sophisticated systems using less expensive controllers than would otherwise be the case."

This portion of applicant's as-filed disclosure is a thorough teaching of the purpose and meaning of a "call...return stack", and makes it quite clear that such stacks are in common use and notoriously well-known in the art. As there are similar and thorough allusions and teachings further in applicant's disclosure upon which the applicant may rely for claim 4, the applicant respectfully requests that the section 112 paragraph 1 and paragraph 2 rejections be withdrawn in response to this paper.

In regard to the rejection of claims 1-20 under 35 U.S.C. 103(a) over Song in view of Bronte, applicant respectfully traverses the Examiner's position.

Applicant's independent claim 1 recites, as filed:

1. A real-time operating system (RTOS) for use with minimal-memory controllers comprising:

a kernel for managing task execution, including context switching; and

a plurality of defined tasks as code sets, individual ones of the tasks having subroutines callable in nested levels for accomplishing tasks;

characterized in that the kernel constrains context switching to occur only task level, rather than allowing context switches at lower sub-routine level.

Applicant urges that a patentable difference with the prior art in this claim is that context switching is constrained to occur only at task level, rather than allowing context switches at lower sub-routine level, and that if *this limitation* is not taught in the art or shown to be obvious from the art, then this independent claim is patentable to the applicant, and all depended claims are then patentable at least as depended from a patentable claim.

After a thorough review of Song, it is clear that Song teaches that minimal processor state information can be saved in a context switch (Fig. 6, 610-618) and later restored (Fig. 6, 626-632) for continued task operation when (and only when) the complete processor state (Col. 1, line 61) need *not* be saved and eventually restored as a requirement for continued task operation. The Examiner uses this to equate to the claims recitation of a "minimal-memory controller". Applicant respectfully points out that management of "processor-state information" during operation is not the same as the amount of memory available to a processor.

Furthermore, Song differentiates (Fig. 6, 608 vs. 610, and elsewhere) between the saving of the Program Return Address and the context of the task. Song considers the saving and restoring of the Program Return address (Fig. 6, 608 and 632) as <u>separate</u> from the saving and restoring of minimal processor state information (Fig. 6, 614 and 630). Song does not include the Program Return Address in the processor state information – he treats it separately as an unavoidable requirement of context switching (which it rightly is).

Song's entire patent is focused on the issues surrounding how to identify and manage the places in a program where one can reduce (*but not eliminate*) the amount of processor state information to successfully suspend and later resume task execution, thus saving time and improving performance.

While Song does not explicitly suggest how much memory (per task) might be required to store minimal processor state information, it must be a non-zero amount, as there would otherwise be no point in saving and restoring this processor state information.

In a conventional RTOS program return address stack & complete processor state information is saved. In Song program return address stack & minimal processor state information is saved. In applicant's invention the program return address only is saved. Applicant's invention eliminates the need for saving *any* processor state information. One challenge that the applicant faced was how to implement the commonly available functionality of an RTOS with the restriction that no processor state information is saved in a context switch.

Importantly, in all of the teaching of Song, the limitation of *"constraining context switching to occur only at task level, rather than allowing context switches at lower sub-routine level"* is not taught or suggested, as admitted by the Examiner at the bottom of page 2 of the instant action.

As to the limitation of *"constraining context switching to occur only at task level, rather than allowing context switches at lower sub-routine level"*, in regard to the secondary reference Bronte, the Examiner states that Bronte "...teaches context switching with a Kernel occurring at the task level". Applicant believes this is the point where the rejection fails completely, because what the Examiner asserts that Bronte teaches, to support the rejection of applicant's claim 1, *is not what the applicant recites in the claim.* Allowing or supporting context switching at the task level is clearly and unarguably not the same as *"constraining context switching to occur only at task level, rather than allowing context switches at lower sub-routine level"*, as claimed. *Constraining* clearly and unarguably means that context switching in the invention defined by applicant's claim 1 can occur <u>only</u> at task level, and this is what the Examiner must find in the art, either in Bronte or another reference, to form an *a priori* rejection. So the standard for an a priori rejection has clearly not been met.

In point of fact, it is Inherent in Bronte's teaching that context switching *must* be supported at *any* level, not just the task level. This is because of his discussion of interrupts. Since an interrupt can occur at any time – at any nested level of calls within a task – the context-switching mechanism of Bronte *must* be able to save the complete nested return program addresses and processor state information of the task for later restoration when the task continues. To accomplish such save-and-restore, memory must be allocated for the nested program return addresses, the processor state information (complete, not minimal, in Bronte's case), and also any other information that must be saved and restored for the proper operation of the task. This latter category includes parameters and auto variables passed on the stack, as would commonly occur in many programming languages (e.g. C).

To summarize, in regard to context switching level, in a conventional RTOS, as well as in Song and Bronte, context switching must be supported at all levels because context switching can occur within or directly due to an interrupt, i.e. completely randomly. In applicant's invention context switching within or directly due to an interrupt is not supported, and therefore context switching may be, and is, constrained as claimed to occur only at a single level, the task level.

Further to the above arguments, on page 3, beginning at line 2, the Examiner states "It would have been obvious ... for the reason of maintaining control of the system." However, general context-switching as taught by Bronte <u>already includes context-switching at any level</u>, including the task level. It does not in any way affect "maintaining the control of the system." It is <u>not</u> obvious that context switching should occur only at the task level. This limitation places considerable restrictions on context switching (e.g. cannot occur directly due to an interrupt, cannot occur within any nested subroutine call, etc.).

Still further, on page 3, the Examiner states "In addition, a kernel is a module that is responsible for process/task management and it makes it easier if data communication is done at a higher level, rather than a lower one." This is simply false. The examiner implies that a kernel's responsibilities are better done at the task level than at a nested level. This is untrue, as conventional RTOS kernels must be able to perform their actions irrespective of the nested level at which the RTOS services are called. For example, assume two tasks each wish to print to a *shared resource*, e.g. a printer. Yet the printer can only accept a complete data stream from one task before accepting another. A typical multitasking application will use the RTOS to manage access to this shared resource by suspending one task until the other is finished with the resource. The context switching to suspend a task will occur within a nested level because the task will use the same interface (e.g. a SendToPrinter() function) to access the shared resource. If SendToPrinter() finds that the resource is currently not available (i.e. in use by another part of the system), the calling task will be suspended (context-switched out) and will only be resumed (context-switched in) when the resource becomes available again. This *encapsulation* provides a means of hiding certain low-level RTOS-controlled system issues (e.g. access to a shared resource) from higher-level processes. *The level at which encapsulation occurs is completely transparent to the user.*

To summarize the differences, in conventional RTOSes (incl. Bronte and Song) kernel services work at all levels. In the instant invention kernel services are restricted to the task level only.

Further yet, also on page 3, the Examiner states that "Song teaches the RTOS operates within a single call...return stack common to all of the defined tasks. This is also false. Song's "soft return address stack" is "uniquely associated with the program being context switched out." (Col 10. line 67, Col. 11 lines 1-2). In other words, each task has its own, private return address stack. This is a common feature of conventional RTOSes and is required whenever context-switching is supported / allowed at any level deeper than the task level. Naturally, this adds to memory requirements.

In conventional RTOS, including Bronte and Song, each task has its own stack for program return addresses. In the instant invention tasks do not use stacks. A single program return address is stored elsewhere. In the instant invention, because of the potentially very large memory requirements of per-task call return stacks, and the inability of severely-memory-limited systems to support them, these are dispensed with entirely. In conventional RTOSes, it is non-trivial to set the size of each task stack. That's because the user has complete latitude in the amount of subroutine nesting, as well as the use of parameters and local auto variables (in C-language implementations, others are similar). Stack overflow is a serious problem at must be avoided. Even when the RTOS provides a means of specifying task stack size on a per-task basis, one usually errs on the conservative size and over-allocates memory for task stacks.

Claim 1 is therefore clearly patentable to applicant over the references cited and applied, taken either singly or in combination, and dependent claims 2-10 are therefore patentable at least as depended from a patentable claim.

Claim 11 is applicant's method claim based on claim 1, and equivalent in limitations. Claim 11 is therefore clearly patentable to applicant over the references cited and applied, taken either singly or in combination, and claims 12-20 are therefore patentable at least as depended from a patentable claim.

As all of the claims as argued above by applicant are clearly patentable over the art of record, taken either singly or in combination, applicant respectfully requests reconsideration, and that the present case be passed quickly to issue. If there are any time extensions due beyond any extension requested and paid with this amendment, such extensions are hereby requested. If there are any fees due beyond any fees paid with the present amendment, such fees are authorized to be deducted from deposit account 50-0534.

Version With Markings to Show Changes Made

No changes to the claims or the specification have been made in this paper, and therefore no changes are required to be shown in this section. Applicant urges that this is completely within all applicable rules, and earnestly requests that no Notice of Non-Compliance be sent, which will simply waste the time of the applicant, his representative, and the Office. This paper is completely compliant with all Rules.

Respectfully Submitted,
Andrew E. Kalman

by [illegible]
Donald R. Boys
Reg. No. 35,074

Donald R. Boys
Central Coast Patent Agency
P.O. Box 187
Aromas, CA 95004
(831) 726-1457